Wireless Networks
https://doi.org/10.1007/s11276-024-03796-0

ORIGINAL PAPER

Exploiting data transmission for route discoveries in mobile ad hoc networks

Xin Yu[1]

Accepted: 6 June 2024

Abstract
On-demand routing protocols discover routes through network-wide searches. Route requests are broadcast to a large number of nodes, and route replies may contain long routes. In this paper, we address the route discovery problem and aim to reduce route discovery overhead. We propose using data packets to discover routes. A source sets a boolean variable in a data packet to be true when it has only one route to the destination. This variable is a new form of a route request. The nodes forwarding the data packet send route replies containing cached routes. To prevent nodes from sending duplicate routes to the source, we define a *forward* list and a *backward* list in the data packet. The node sending a route reply records route diverging and converging information about the route in the route reply. Subsequent nodes use the information in the data packet to decide whether to send a route reply. Our algorithm reduces route discovery latency and discovers routes shorter than or having the same length as the *active* data path. Due to these shorter routes, it reduces the total size of route requests and route replies significantly. Routing overhead increases slowly as mobility or network load increases. Our algorithm is independent of node movement. It improves packet delivery ratio by 15% and reduces latency by 54% for the 100-node networks at node mean speed of 20 m/s.

Keywords Routing protocols · Mobile ad hoc networks · Route discovery problems · Broadcast storms · Directed route discoveries · Mobility independent · Highly mobile scenarios · Space communications · Delay-tolerant networks · Scalability issues

1 Introduction

In a mobile ad hoc network (MANET), on-demand routing protocols discover routes through network-wide searches. To find a route to a destination, a source broadcasts a ROUTE REQUEST. The node receiving a ROUTE REQUEST sends a ROUTE REPLY to the source if it has a cached route and rebroadcasts the packet if it does not have a cached route to the destination. ROUTE REQUESTS are forwarded to a large number of nodes, and ROUTE REPLIES may contain long routes. A straightforward broadcasting by flooding results in serious redundancy, contention, and collision. This problem is called broadcast storms [1]. Routing packets consist of ROUTE REQUESTS, ROUTE REPLIES and ROUTE ERRORS. Routing overhead is the total number or the total size of routing packets transmitted. For packets sent over multiple hops, *each* transmission of the packet is counted as one transmission [2]. ROUTE REQUESTS and ROUTE REPLIES are the major source of routing overhead. As mobility increases, routes break more frequently. Network-wide searches incur more routing overhead and packet losses caused by MAC (Media Access Control) collisions. In this paper, we address the route discovery problem in mobile ad hoc networks and aim to reduce route discovery overhead, namely the number and the size of route discovery messages. Route discovery overhead is the total size of ROUTE REQUESTS and ROUTE REPLIES. The larger a packet is, the higher probability packet collisions will occur with. Large amount of routing packets cause severe wireless interference and packet losses, degrading network throughput and capacity.

To address the route discovery problem, previous studies mainly focused on restricting search space. We describe the most well- known two pieces of work first and will discuss more related work shortly. Ko and Vaidya [3] suggested an

✉ Xin Yu
yu_xin2014@hotmail.com

[1] Sen Pu Software Corporation, Jinan, Shandong Province, China

Published online: 31 August 2024

approach to reduce the overhead of route discovery by utilizing location information. Such information will be obtained using global position system (GPS). They presented two Location-Aided Routing (LAR) protocols for route discovery. The LAR protocols aim to reduce the search space for a desired route, and limiting the search space leads to fewer route discovery messages. The protocols use location information to limit a search to a request zone, which is determined based on the past location of the destination and its speed. However, this work requires GPS. Castaneda and Das [4] proposed a query localization technique, which uses prior routing histories to estimate a small region with high probability of finding the destination. A smaller region results in lower routing overhead, but a route within the region may not exist. If a route search fails, another route discovery with a larger region increases route discovery latency. Thus, there is a trade-off between routing overhead and route discovery latency. Subsequent research mainly used probabilistic methods [5–7]. Compared with using flooding, the probabilistic methods reduce route discovery messages significantly, but their performance depends on whether the time-distance correlation holds. The authors assume mobility processes are homogenous, but node movement is fully random in reality. Moreover, these approaches cannot guarantee that a route to the destination can be found.

In contrast to previous studies, our goal is to reduce the total number of network-wide route searches or broadcastings. We present an algorithm called *directed route discovery* (DRD) for the Dynamic Source Routing protocol (DSR) [8], and this algorithm discovers routes through data packets. The intuition behind this idea is that each source node continuously sends data packets to the destination. Initially, a source discovers routes through a network-wide search. It caches routes sent back through ROUTE REPLIES. When it has only one route to the destination, it piggybacks a ROUTE REQUEST on a data packet, which is a *boolean* variable. The nodes forwarding the data packet send ROUTE REPLIES using cached routes. These ROUTE REPLIES contain the routes shorter than or having the same length as the current route being used. Thus, the protocol reduces the total size of ROUTE REPLIES, since a ROUTE REPLY in DSR contains a source route from the source to the destination. As long as an intermediate node has a cached route that satisfies the condition, the algorithm avoids a route search.

The nodes forwarding the data packet should avoid sending duplicate routes to a source. In DSR, each packet contains a source route from the source to the destination. A ROUTE REQUEST contains a sequence of nodes that have been traversed. If a node receiving a ROUTE REQUEST has a cached route to the destination, it uses the route to construct a source route and sends a ROUTE REPLY to the source. The node sending a ROUTE REPLY does not propagate the ROUTE REQUEST to its neighboring nodes, and thus no ROUTE REPLIES contain the same route. In our method, a ROUTE REQUEST is a *boolean* variable in a data packet. When it is *true*, it indicates that a source wants to find routes to a destination. When a data packet contains a ROUTE REQUEST, several nodes on the *active* data path may send the same route to the source.

To address this problem, we define a *forward* list and a *backward* list in a data packet. The node sending a ROUTE REPLY records route diverging and converging information about the cached route in the ROUTE REPLY. Diverging information refers to the first node pair where a cached route diverges from the source route. Converging information refers to the first node pair where a cached route and the source route converge. Each node on the data path uses the information in the data packet and two rules to decide whether to send a ROUTE REPLY.

The algorithm has four desirable properties. First, it reduces route discovery latency because routes are obtained before the last route breaks. Second, it is independent of node movement. The intermediate nodes will send ROUTE REPLIES to a source no matter how fast the source and the destination move. Third, it finds shorter routes and thus reduces packet delivery latency. Finally, it reduces the size of both ROUTE REPLIES and ROUTE REQUESTS. In DSR, the size of a ROUTE REQUEST depends on the number of hops traversed. In our method, a source needs to broadcast a ROUTE REQUEST when no ROUTE REPLY is sent by intermediate nodes. We found that the total size of ROUTE REQUESTS is significantly reduced. Since most of found routes are shorter than the current route, the nodes close to the source are in the vicinity of the destination and have cached routes to the destination. They send ROUTE REPLIES, and ROUTE REQUESTS are not broadcast further.

We show that the algorithm significantly improves packet delivery ratio and reduces packet delivery latency. Packet delivery ratio describes the loss rate that will be seen by the transport protocols and characterizes both the completeness and correctness of the routing protocol [2]. For the 100-node networks, it improves packet delivery ratio by 15% and reduces packet delivery latency by 54%. It reduces packet delivery latency by 48% and 45% for the 150- node and the 200-node networks, respectively. Simulation results show that 80% of routing overhead results from route discoveries. The algorithm reduces route discovery overhead by 78% and 81% for the 150-node and 200-node networks, respectively. It reduces the total size of ROUTE REQUESTS by 75% for the 150-node networks and 79% for the 200-node networks. Under high network load, such as 40 flows, it improves packet delivery ratio by 22% and reduces packet delivery latency by 46% for the 100-node networks at node mean speed of 10 m/s.

This paper makes two contributions. First, we present a new route discovery algorithm for one of the most popular on-demand routing protocols in mobile ad hoc networks. Second, due to finding shorter routes, the algorithm reduces the length of the routes traversed by ROUTE REPLIES, and ROUTE REQUESTS sent through broadcast are handled by the nodes close to a source. The main goal of the algorithm is to reduce network-wide route searches. For the 100-node networks, it reduces route searches by 17%. For the 150-node and 200-node networks, it reduces route searches by 12% and 17%, respectively. Because of the above three features, our algorithm reduces route discovery overhead significantly. Moreover, routing overhead increases slowly as mobility or network load increases. Routing overhead measures the scalability of a routing protocol, the degree to which it will function in congested or low-bandwidth environments, and its efficiency in terms of consuming node battery power [2].

Li et al. [9] show that traffic pattern determines whether per node capacity scales to large networks, and the average distance between sources and destinations must remain small as the network grows. We show that the locality of routing packets is critical for the scalability of on-demand routing protocols. However, as network load increases, both packet delivery ratio and delivery latency degrade rapidly because of a large amount of wireless interference and packet collisions. Making network throughput or capacity degrade slowly with respect to network load is not solved.

The rest of this paper is organized as follows. In Sect. 3, we give an overview of DSR. In Sect. 4, we describe the directed route discovery protocol. We present simulation methodology and simulation results in Sects. 6 and 7, respectively. In Section 2, we discuss related work. Finally, we conclude in Sect. 8.

2 Related work

Castaneda and Das [4] proposed query localization protocols in which a query message is flooded in the neighborhood of the prior route. The message includes a set of nodes on the prior route, and is forwarded by the nodes for which the source route in the packet has at most k nodes not on the route. If the region is too small, a route may not exist, and another route discovery with incremented k will be needed, which increases route discovery latency. A larger region increases routing overhead. Thus, there is a trade-off between latency and routing overhead. In contrast, our protocol reduces both packet delivery latency and routing overhead.

In the work of Ferriere et al. [6], successive searches advance toward the destination. A source searches for a node that encountered the destination more recently. This procedure continues until the destination is found. The intuition is that if a node encounters a destination more recently than another node, the former is probably closer to the destination. However, the performance of the algorithm depends on whether the time-distance correlation holds. If mobility processes are heterogeneous, the relationship between encounter age and distance becomes noisy.

Haas et al. [5] proposed a gossiping-based approach to reduce the overhead of flooding commonly used in on-demand protocols. Each node forwards a message with some probability. The results in percolation theory [10] show that gossiping exhibits a certain type of bimodal behavior. Let the gossip probability be p. Then, in sufficiently large graphs, there are fractions $\theta^S(p)$ and $\theta^R(p)$ such that the gossip quickly dies out in 1—$\theta^S(p)$ of the executions and, in almost all of the fraction $\theta^S(p)$ of the executions where the gossip does not die out, a fraction $\theta^R(p)$ of the nodes get the message. Moreover, $\theta^R(p)$ is close to 1 in many cases. Thus, in almost all executions of the algorithm, either hardly any nodes receive the message, or most of them do. They show that using gossiping probability between 0.6 and 0.8 suffices to ensure that almost every node gets the message in almost every execution. They also show that, by using appropriate heuristics, the gossiping protocol saves up to 35% message overhead compared with flooding. However, with gossiping, a source may not always discover the shortest routes. The gossiping protocol may not guarantee that a route to the destination is found, either because the destination is missed due to inappropriate gossiping probability choices or because the physical layer broadcast is sent with certain probability.

Beraldi [7] presented a probabilistic protocol called polarized gossiping for path discovery in mobile ad hoc networks. The gossiping probability of a node is variable and high enough only for sustaining the spreading process towards the destination. It is determined by the difference between its proximity to the destination and the proximity to the destination of the node from which the message was received. The proximity is obtained using periodic beacons. The polarized gossip algorithm contains a polarizing node, n^*, and two gossiping probabilities, pF and pB. If node i receives a message for the first time and from node j, it forwards the message with probability pF if i is closer than j to the destination and with probability pB otherwise. The author shows that the protocol saves up to 80% of broadcast transmissions compared with flooding. The above two probabilistic protocols are more scalable than the protocols using flooding, but their performance also depends on whether the time-distance correlation holds.

Hui et al. [11] studied the data forwarding problem for Pocket Switched Networks (PSN), a type of Delay Tolerant

Networks [12]. A PSN allows humans to communicate without network infrastructure. They exploited two social metrics, namely centrality and community, using real human mobility traces. They designed and evaluated BUBBLE, a novel social-based forwarding algorithm, that utilizes the two metrics to improve message delivery performance. Messages bubble up and down the social hierarchy, based on the observed community structure and node centrality, together with explicit label data. However, there are several issues for this algorithm. First, message delivery ratio is lower than the routing protocols designed for mobile ad hoc networks. For example, delivery success ratio is less than 0.5 when the time TTL of the messages (the maximum time a message can stay in the system after its creation.) is one week, and less than 0.6 when the time TTL is three weeks. As shown in Fig. 5, packet delivery ratio is 0.85 for DSR with DRD and 0.8 for DSR, for the 150-node networks at pause time 0 s and with node mean speed of 10 m/s. Second, packet delivery latency of the BUBBLE algorithm is high. For example, if the time TTL is one day, delivery success ratio is only less than 0.2 on *Reality* dataset because the *Reality* network is very sparse. In contrast, packet delivery latency is 0.42 s for DSR and 0.22 s for DSR with DRD for the 150-node networks at pause time 0 s. It is unclear how TCP performs on top of such social-based forwarding algorithms. Finally, node moving speed is very low in a PSN because nodes are devices carried by humans. We believe that social-based forwarding does not apply to mobile ad hoc networks with mild mobility, such as node mean speed of 5 m/s, or high mobility.

Lee et al. [13] presented the Whirlpool Routing Protocol (WARP) for sensor networks, which efficiently routes data to a node moving within a static mesh. The key insight behind WARP is that, when a destination moves, the existing distance vector tree can quickly find its new location. WARP uses the existing topology to search around the destination's old location for nodes that still have routes. When it finds such a node, it quickly repairs the local topology. WARP uses speculative routing to search for a destination's new location. Speculative routing sends packets along a spiral trajectory around the last known position of the destination. Similar to query localization [4], WARP limits the range over which nodes search for a route. However, as discussed, WARP becomes inefficient when the destination moves very fast. Both the path length and routing overhead increase dramatically, and spiral whirlpool packets fail to find the new location of the destination.

Ladas et al. [14] presented Multipath-ChaMeLeon (M-CML) as an update of the existing ChaMeLeon (CML) [15] routing protocol. The concept of multi-path is that a source maintains multiple paths to a destination and distributes load among these paths [16, 17]. CML is a hybrid protocol designed for mobile ad hoc networks, which adapts its routing behavior according to network size. For small networks, CML routes data proactively using the OLSR (Optimized Link State Routing) [18] protocol, and for large networks, it utilizes the reactive AODV [19] protocol. They implemented the multi-path approach on the proactive phase of CML. M-CML extends OLSR so as to calculate multiple paths based on the Expected Transmission Count (ETX) [20]. To reduce the generation of multiple duplicated messages produced, M-CML implements a simplified approach by sending data to the two most optimal disjoint next addresses. They show that M-CML's multipath routing combined with ETX significantly reduces end-to-end delay compared with OLSR. Our algorithm finds shorter routes when the source is sending data packets to the destination but does not send duplicated packets to several paths. Although it maintains multiple paths, it is different from multi-path routing.

Tang et al. [21] studied the problem of on-demand route discovery in asynchronous duty-cycling sensor networks and presented four optimizations: *Delayed Selection*, *Duty-Cycled Selection*, *Reply Updating* and *Adaptive Backoff*. They show that with only simple changes made at the MAC or network layers, their optimizations enabled nodes to improve significantly discovered routes while reducing route discovery latency and node energy consumption. These optimizations can find routes that were only 0.2% longer than the theoretical shortest routes or routes with an ETX only 9% larger than the ETX of the theoretical optimal ETX routes. However, this work focused on asynchronous dutycycling sensor networks and does not aim to reduce route discovery overhead in mobile ad hoc networks.

Current protocols for sensor networks mainly support multihop *upward* traffic from many sensors to a collection point; however, the scenarios involving low-power wireless devices increasingly require support for *downward* traffic. For example, a controller needs to issue actuation commands based on the monitored data. The IETF Routing Protocol for Low-power and Lossy Networks (RPL) [22] deals with both traffic patterns. Istomin et al. [23] observed that a simple dissemination-based flooding protocol systematically provided near-perfect reliability even in scenarios where the routing-based approach of ContikiRPL performed unacceptably. As a result, they tackled the routing problem by extending RPL with two extremes. At one extreme, they retained the route-based operation of RPL and added techniques that were neglected by popular implementations. At the other extreme, they rely on flooding as the main networking primitive. In order to send commands to the monitored area, the root needs to find routes to a small group of sensors; this is a route search problem. The authors show that a carefully balanced mix of routing and flooding strikes performance tradeoffs not achievable by either approach.

Table 1 Comparison of seven popular methods for route discovery in mobile ad hoc networks

Name	Advantages	Limitations
LAR	It reduces route discovery messages using request zone	GPS is required
Query localization	A smaller search region causes lower routing overhead	If the region is too small, a route may not exist
Gossip-based routing	It saves up to 35% message overhead	It may not guarantee that a route is found
Bubble rap	Messages bubble up and down the social hierarchy	Message delivery ratio is lower and latency is high
Whirlpool routing	It limits the range over which nodes search for a route	It is inefficient when the destination moves fast
CGR	Contact plans are determined in advance	It has scalability issues and TCP cannot be used
RPL	They combined both routing and flooding	Flooding is used as the main networking primitive

The energy consumption problem is severe for large-scale systems that involve several thousands servers. Zakarya et al. [24] discussed the energy consumption and performance problems of large-scale systems and presented several taxonomies of energy and performance aware methodologies. To solve the route discovery problem, our goal is to find routes as fast as possible while incurring as few routing packets as possible. These routing packets cause interference, collisions, and retransmissions, consuming limited bandwidth and node energy. This work fits into the broad range of Zakarya's research of energy efficiency and performance problems.

Araniti et al. [25] presented an overview, enhancements, and the performance for Contact Graph Routing [26] in DTN (Delay-Tolerant Networking) [27] space networks. They said:" Satellite systems already have to cope with difficult communication challenges:long round trip times (RTTs); the likelihood of data loss due to errors on the communication link; possible channel disruptions; and coverage issues at high latitudes and in challenging terrain. These problems are magnified in space communications characterized by huge distances among networks nodes, which imply extremely long delays and intermittent connectivity. At the same time, a space communication system must be reliable over time due to the long duration of space missions. Moreover, the importance of enabling Internet-like communications with space vehicles is increasing." As described by Wyatt et al. [28], DTN was originally designed to meet the emerging need to provide capable network services for space flight operations. It runs directly over link-layer protocols, taking the place of the IP network protocol when necessary. Route computation is based on a schedule of planned contacts, and forwarding at each router is store-and-forward rather than immediate. In NASA's (National Aeronautics and Space Administration) EPOXI mission [28], the routing protocol used is CGR, which computes dynamic routes through time-varying network topology of scheduled communication contacts in a DTN network. The fundamental features of space networks are interplanetary distances and high mobility of spacecrafts; topology changes occur more rapidly than they can be reported.

The DTN architecture introduces an overlay protocol that interfaces with either the transport layer or lower layers. Each node can store information for a long time before forwarding it. "Approaches that assume minimal accurate network state information have historically been considered 'opportunistic' while those that assume complete network state information are regarded as 'deterministic."' [29] Opportunistic approaches rely on the exchange of infrastructure and/or in-networks measurements in a timely manner to support on-demand calculations of routes. Such approaches often use a replication-based strategy and are suitable for networks with high mobility and nearly random contact establishment. In scheduled DTNs, contact plans are determined based on the estimations of future episodes of communication. Madoery et al. [29] said:"Whether kept in centralized planning node or distributed to all nodes, the contact plan is used by algorithms such as CGR to derive efficient routes." In addition to knowing contact plans in the planning stage, CGR faces the scalability problem, namely computing routes through a contact plan comprising millions of contacts in large space networks.

We believe that mobile ad hoc networks are important for space networks; spacecrafts are mobile nodes with fast moving speeds and intermittent connectivity. The routing protocols for ad hoc networks will address the high mobility of spacecrafts and the scalability issues. Our protocol reduces route discovery latency and packet delivery latency. Both metrics are critical for space networks. The algorithm applies to space networks where contacts plans are not known ahead of time and the network is arbitrarily large. The feasibility of mobile ad hoc networks for space communications is yet to be validated and proved by NASA and other space agencies. The reasons why such networks have not been considered for space communications are also worthy of investigations. DTN is a concept, and the reality of space communications is that nodes are highly mobile. When data volume is large, store-and-forward mode will encounter problems too.

For easy comparison with the most significant related studies, we summarize seven pieces of previous work that have been widely cited in a table (Table 1), namely the LAR protocol [30], query localization techniques, gossip-based routing, social based forwarding (bubble rap) for PSN [31], whirlpool routing protocol (WARP) for sensor networks, Contact Graph Routing for space networks [26], and efficient support for *downward* traffic in RPL [23].

3 overview of DSR

DSR consists of two on-demand mechanisms: Route Discovery and Route Maintenance. When a source node wants to send packets to a destination to which it does not have a route in its cache, it initiates a Route Discovery by broadcasting a ROUTE REQUEST. The node receiving a ROUTE REQUEST checks whether it has a route to the destination in its cache. If it has, it sends a ROUTE REPLY to the source node including a source route, which is the concatenation of the source route in the ROUTE REQUEST and the cached route. Otherwise, the node adds its address to the source route in the ROUTE REQUEST and rebroadcasts the packet. When a ROUTE REQUEST reaches the destination, the destination sends a ROUTE REPLY containing the source route to the source. Each node forwarding the ROUTE REPLY caches the route starting from itself to the destination. When the source receives a ROUTE REPLY, it caches the source route. A source may learn multiple routes through a Route Discovery and will cache all found routes.

In Route Maintenance, a node forwarding a packet is responsible for confirming that the packet has been delivered to the next hop. If no acknowledgement is received after the maximum number of retransmissions, the forwarding node assumes that the next hop is unreachable and sends a ROUTE ERROR to the source node. Each node receiving a ROUTE ERROR removes from its cache all routes containing the broken link.

Besides Route Maintenance, DSR uses two mechanisms to remove stale routes. First, a source piggybacks on the next ROUTE REQUEST the last broken link information called a GRATUITOUS ROUTE ERROR. Second, DSR uses heuristics: a small cache size for path caches with FIFO (First-In-First-Out) and adaptive timeout mechanisms for link caches [32], where link timeouts are chosen based on observed link usages and breakages. However, topology changes are fast and unpredictable, and adaptive timeout mechanisms may keep stale routes and remove good ones. In order to remove stale routes quickly, Yu and Kedem presented a new cache structure called *cache table* and a distributed cache update algorithm for DSR [33]. The earliest version of DSR uses path caches, and cache size is fixed. A small cache size causes DSR to fail to scale to large networks. We studied the impact of the cache update algorithm and a cache table without capacity limit on the scalability of DSR in [34]. Details about how they handled topology changes and link failures are described in [33].

4 Directed route discovery

In this section, we first present the motivations of this work and give an overview of the algorithm. We then present two examples in order for the readers to quickly understand its operations. Later, we describe the algorithm step by step based on pseudo code and provide the data flow diagrams. Finally, we discuss several implementation details and the comparison with SLR (Source routing with Local Recovery) [35] with its limitations.

4.1 Motivations and concepts

Mobile ad hoc networks apply to situations like battlefields or disaster areas where networks need to be deployed immediately but base stations are not available [1]. Although MANETs have been studied for more than two decades, the route discovery problem has not been solved. The major method for finding a route is broadcastings or pure flooding. Due to mobility, broadcastings are expected to be performed frequently because a route breaks when two nodes move out of the transmission range of each other. Ni et al. [1] observed that serious redundancy, contention, and collision could exist if flooding is done blindly. First, because a physical location may be covered by the transmission range of several hosts, many rebroadcasts are redundant. Second, heavy contention could exist because rebroadcasting hosts are probably close to each other. Third, packet collisions are more likely to occur because the RTS (Request-To-Send)/CTS (Clear-To-Send) dialogue is inapplicable, and the timing of rebroadcasts is highly correlated. The problems associated with flooding are called broadcast storms [1].

When broadcastings are used, ROUTE REPLIES may contain long routes. A source node uses these long routes until they are broken or evicted by FIFO policy when DSR uses path caches. The packet size of such ROUTE REPLIES is large, and these long routes lead to high packet delivery latency too. Moreover, ROUTE REQUESTS are broadcast to the entire network and will be dropped until they reach the maximum number of hops. *Routing overhead* is the total number or the total size of routing packets transmitted. *Route discovery overhead* is the total size of ROUTE REQUESTS and ROUTE REPLIES. In this paper, we aim to reduce route discovery overhead. The solution is to reduce the total number of network-wide searches or broadcastings.

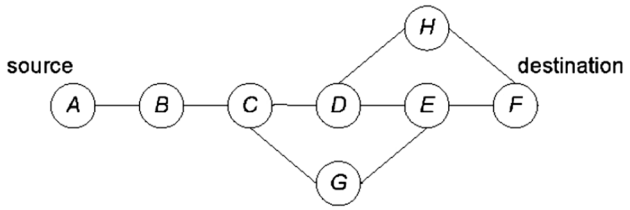


Fig. 1 An Example of the Directed Route Discovery Protocol

4.2 Overview

In on-demand routing protocols, a source node needs to find routes before it starts data transmission. Found routes are stored in a route cache. The key idea of the DRD algorithm is to use data packets to find routes so as to avoid broadcasting ROUTE REQUESTS. To achieve this goal, the problem to be solved is that we should prevent intermediate nodes from sending duplicate routes to the source. As shown in Fig. 1, when source *A* has only one route *ABCDEF* to the destination, node *F*, it piggybacks a ROUTE REQUEST on a data packet to node *F*. Node *B* has a route *BCGEF* and sends a ROUTE REPLY containing route *ABCGEF*. Node *C* has a route *CGEF* and also sends a ROUTE REPLY containing route *ABCGEF*. Node *E* has a route *EGCBA* and sends a ROUTE REPLY containing route *ABCGEF*. The node forwarding the data packet does not know what routes have been sent to the source by the preceding nodes.

We define a *forward* list and a *backward* list to capture the minimum information that identifies a cached route. A *forward* list contains the first diverging node pairs of the routes starting from the current node. A diverging node pair refers to a node and its next hop where the route in a ROUTE REPLY diverges from the source route. For example, in Fig. 1, the source route is *ABCDEF*, which is contained in each data packet. Node *C* and node *G* are the first diverging node pair of route *BCGEF*. If the first diverging node pairs of two routes are different, the routes must be different. A *backward* list contains the first converging node pairs of the routes starting from the current node. A converging node pair refers to a node and its previous hop where a cached route and the source route converge. For example, node *E* and node *G* are the first converging node pair of route *BCGEF*. The first converging node pair is used to identify a route to the source node. In DSR, a node may use a route to the source node or the destination node to construct a source route and send a ROUTE REPLY containing this source route. The node where two routes diverge or converge is called an *anchor*.

After receiving a data packet, the node sending a ROUTE REPLY adds a diverging node pair and/or a converging node pair to the data packet. If a node uses a route to the destination node to send a ROUTE REPLY, it adds the first diverging node pair to the *forward* list and the first converging node pair to the *backward* list. If it uses a route to the source node to send a ROUTE REPLY, it adds the first converging node pair to the *backward* list.

In order to decide whether to send a ROUTE REPLY, the nodes forwarding the data packet containing a ROUTE REQUEST use two rules. The first rule is that, the node uses a route to the source node to send a ROUTE REPLY if the route does not contain any converging node pair in the *backward* list. The second rule is that, the node uses a route to the destination node to send a ROUTE REPLY if the *forward* list does not contain the first diverging node pair of the route. The two rules ensure that the intermediate nodes on the current route do not send duplicate routes to the source node.

We show a simple example in Fig. 1. Node *B* uses route *BCGEF* to send a ROUTE REPLY containing route *ABCGEF*. It adds node *C* and node *G* to the *forward* list, and node *E* and node *G* to the *backward* list. When node *C* receives the data packet, according to the second rule, it will not use route *CGEF* to send a ROUTE REPLY, since the *forward* list contains the first diverging node pair of this route. Node *C* will use route *CDHF* to send a ROUTE REPLY containing route *ABCDHF*. It adds node *D* and node *H* to the *forward* list, and node *F* and node *H* to the *backward* list. Thus, according to the second rule, node *D* will not use route *DHF* to send a ROUTE REPLY. When node *E* receives the data packet, according to the first rule, it will not use route *EGCBA* to send a ROUTE REPLY, since this route contains a converging node pair in the *backward* list. Similarly, node *F* will not send a ROUTE REPLY containing either route *ABCGEF* or route *ABCDEF*.

4.3 Examples

We show two examples in more details. The first example is shown in Fig. 2. When source *A* has only one route *ABCDEF* to destination *F*, it sets a boolean variable called *has_rreq* in a data packet to be *true*. This variable indicates whether a source wants to find routes to a destination. Node *B* has a cached route *BGHEF* and sends a ROUTE REPLY containing route *ABGHEF*. Before sending the ROUTE REPLY, node *B* adds the first converging node pair, node *E* and node *H*, to the *backward* list, so that node *E* and the nodes after it do not send a ROUTE REPLY containing the same route, *ABGHEF*. Node *B* forwards the data packet to the next hop.

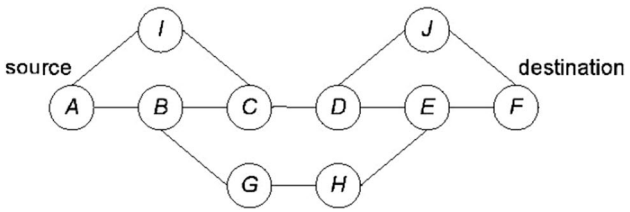


Fig. 2 Example 1 of the Directed Route Discovery Protocol

Node *C* has a cached route to the source, *CIA*. The route does not contain any converging node pair in the *backward* list, and thus node *C* uses the route to send a ROUTE REPLY. Node *C* adds the first converging node pair, itself and node *I*, to the *backward* list, so that the nodes after it do not use route *CIA* to send a ROUTE REPLY. It also has a route to the destination, *CDJF*. The first diverging node pair of this route is not in the *forward* list, and thus node *C* uses this route in a ROUTE REPLY. It adds node *D* and node *J* to the *forward* list, so that node *D* does not use route *DJF* to send a ROUTE REPLY. It also adds node *F* and node *J* to the *backward* list. Node *C* constructs a source route *AICDJF* and sends a ROUTE REPLY to node *A*.

Node *D* has route *DCIA* and route *DJF*, but cannot use both of them. Route *DCIA* contains a converging node pair, node *C* and node *I*, in the *backward* list, and the first diverging node pair of route *DJF* is in the *forward* list. Node *E* has route *EHGBA*, but the route contains a converging node pair in the *backward* list, node *E* and node *H*. Thus, node *E* does not send a ROUTE REPLY containing route *ABGHEF*. Node *F* does not send a ROUTE REPLY containing either route *AICDJF* or route *ABGHEF*. The first converging node pair of route *FJDCIA* contains node *F* and node *J*, and it is in the *backward* list, which means the preceding nodes have sent a route containing this link. Node *F* also has a route *FEHGBA*, and the first converging node pair contains node *E* and node *H*. The *backward* list contains this link.

In summary, when a node uses a route to the source node to send a ROUTE REPLY, it adds the first converging node pair to the *backward* list. When it uses a route to the destination node, it adds the first diverging node pair to the *forward* list and the first converging node pair to the *backward* list.

In the second example shown in Fig. 3, we show the case where the algorithm does not attempt to find all cached routes. Source node *A* starts a *directed route discovery*. Node *B* has route *BCHEFG*. It adds node *C* and node *H* to the *forward* list, and node *E* and node *H* to the *backward* list. It sends a ROUTE REPLY containing route *ABCHEFG*. Assume that node *C* only has route *CHJFG*. It cannot use the route because the *forward* list contains the first diverging node pair, node *C* and node *H*, although the route is different from the route sent by node *B*.

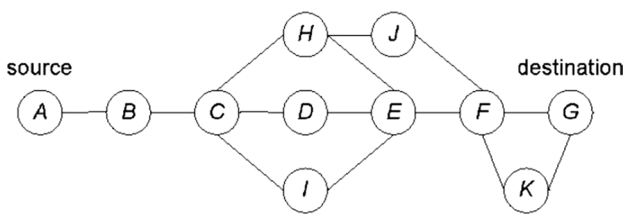


Fig. 3 Example 2 of the Directed Route Discovery Protocol

Assume that node *C* has route *CIEFKG*. Since it is the first anchor, it adds only the first converging node pair, node *E* and node *I*, to the *backward* list. Thus, node *E* and the nodes after it do not send a route containing link *IE*. They will not send route *ABCIEFKG* to the source. If any of these nodes has the same route, it must contain the converging node pair added by node *C*. The first rule allows an intermediate node to decide whether a route to the source was used by the preceding nodes to construct a source route.

4.4 Algorithm description

The pseudo code for the algorithm is shown in Algorithm 1. When a source has only one route to the destination, it sets a *boolean* variable called *has_rreq* in the source route header of a data packet to be *true*. This variable is a new form of a ROUTE REQUEST. It also maintains an integer variable called *request_status* for each destination locally, which is set to 1 when the source piggybacks a ROUTE REQUEST on a data packet. When the source receives a ROUTE REPLY, it sets the corresponding *request_status* to be 0, so that it can piggyback another ROUTE REQUEST when only one route is available. If none of the intermediate nodes sends a ROUTE REPLY, *request_status* for the destination will be 1 till the last route breaks. The source does not initiate another route discovery using a data packet during this period. *Request_status* will be set to 0 when the source broadcasts a normal ROUTE REQUEST. We add three variables to the source route header of a data packet: *forward* list, *backward* list, and *has_rreq*. The type of both lists is array.

The node forwarding the data packet checks whether it has a route to the source or the destination. It first checks whether it has a route to the source (lines: 7–19). This route should be shorter than or have the same length as the current route to the source. The node checks the *backward* list to decide whether to send a ROUTE REPLY. It will use the route if it does not contain any converging node pair in the *backward* list. If the route was sent by a preceding node, that node had added the first converging node pair to the *backward* list. If the node uses the route, it will add the first converging node pair of the route to the *backward* list.

The node then checks whether it has a route to the destination (lines: 20–37). This route should be shorter than or have the same length as the current route to the destination. It uses the route to send a ROUTE REPLY if the first diverging node pair is not in the *forward* list. If the route has been sent, the *forward* list should contain the first diverging node pair. If the node uses the route, it will add the first diverging node pair to the *forward* list. As described in Sect. 4.2, we call the node where two routes diverge or converge an *anchor*. Due to adding the first diverging node pair to the data packet, the nodes between the current node and the first anchor do not send the same route to the source. The node also adds the first converging node pair to the *backward* list. According to the first rule, the nodes after the second anchor

do not send a route containing the node pair. If the current node is the first anchor, it will add only the first converging node pair to the *backward* list.

will set *has_rreq* in the source route header to be *false*, so that subsequent nodes on the route will not send ROUTE REPLIES. It will also remove all entries in both lists and forwards the data packet to the next hop (lines: 43-48). A

Algorithm 1 Directed Route Discovery(DRD)

```
Input:
    PACKET p: a data packet, ID net_id: the current node,
    srh: the source route header of data packet p,
    PATH upstream, PATH downstream,
    PATH backward_new, PATH forward_new,
    boolean use_backward, boolean use_forward,
    ID next_hop: the next hop of an anchor,
    ID previous_hop: the previous hop of an anchor
Output:
    backward_list: the converging node pairs,
    forward_list: the diverging node pairs
 1: if srh.has_rreq and p.src == p.route[0] then
 2:   upstream = the upstream nodes of p.route;
 3:   upstream.reverse();
 4:   downstream = the downstream nodes of p.route;
 5:   use_backward = false;
 6:   use_forward = false;
 7:   if findNewRoute(p.src, upstream, backward_new) and
      backward_new.length ≤ upstream.length then
 8:     first_anchor = the first anchor on backward_new;
 9:     next_hop = the next hop on backward_new;
10:     use_backward = true;
11:     for each entry e ∈ backward_list do
12:       if link(e.anchor, e.next_hop) ∈ backward_new then
13:         use_backward = false
14:       end if
15:     end for
16:     if use_backward then
17:       backward_list = backward_list ∪ (first_anchor, next_hop)
18:     end if
19:   end if
20:   if forward_list.length < MAX_LIST_ENTRY and
      findNewRoute(p.dest, downstream, forward_new) and
      forward_new.length ≤ downstream.length then
21:     first_anchor = the first anchor on forward_new;
22:     next_hop = the next hop on forward_new;
23:     use_forward = true;
24:     if (first_anchor, next_hop) ∈ forward_list then
25:       use_forward = false
26:     end if
27:     if use_forward then
28:       second_anchor = the second anchor on forward_new;
29:       previous_hop = the previous hop of the second anchor;
30:       if net_id == first_anchor then
31:         backward_list = backward_list
            ∪ (second_anchor, previous_hop)
32:       else
33:         forward_list = forward_list
            ∪ (first_anchor, next_hop);
34:         backward_list = backward_list
            ∪ (second_anchor, previous_hop)
35:       end if
36:     end if
37:   end if
38:   for each entry e ∈ forward_list do
39:     if e.anchor == net_id then
40:       forward_list = forward_list \{e};
41:     end if
42:   end for
43:   if backward_list.length == MAX_LIST_ENTRY then
44:     srh.has_rreq = false;
45:     srh.backward_len = 0;
46:     srh.forward_len = 0;
47:   end if
48:   sendOutPacketWithRoute(p);
49:   if use_backward then
50:     upstream = backward_new.reverse()
51:   else
52:     upstream = upstream.reverse()
53:   end if
54:   upstream = upstream.subpath(0, upstream.length -2);
55:   if use_forward then
56:     upstream.appendPath(forward_new)
57:   else
58:     upstream.appendPath(downstream)
59:   end if
60:   if use_backward or use_forward then
61:     rrep.srh.route = upstream;
62:     rrep.src = net_id;
63:     rrep.dest = p.src;
64:     sendOutPacketWithRoute(rrep)
65:   end if
66: end if
```

If the current node is an anchor in the *forward_list*, it will remove the entry because subsequent nodes do not need this information (lines: 38-42). If the number of entries in the *backward* list reaches the maximum value, the current node

source route in DSR contains at most 16 nodes [8], and the maximum number of entries of both lists is set to be 5.

The node concatenates the route to the source and the route to the destination found from the cache as a source route and sends

a ROUTE REPLY (lines: 49–66). If the node uses only the route to the destination, it will concatenate the upstream nodes of the current route and the cached route as a source route. If it uses only the route to the source, it will concatenate the route and the downstream nodes of the current route as a source route.

For the first rule, it is insufficient that the *backward* list does not contain the first converging node pair. If a node uses a route to the destination, it will add only the first converging node pair to the *backward* list. However, the route may have several converging node pairs. The nodes after those node pairs may have a route to the source, which was sent to the source before. But the first converging node pair of the route, which is the last converging node pair of the route sent by a preceding node, was not in the *backward* list. Therefore, this rule requires that a route to the source do not contain *any* converging node pair in the *backward* list.

If the data packet containing a ROUTE REQUEST encounters a link failure, the node detecting the link failure will piggyback a notification on a ROUTE ERROR. When the source receives the ROUTE ERROR, it sets *request_status* for that destination to be 0 so that the source will start a new route discovery when only one route is available in its cache.

When a data packet is salvaged after a link failure, the salvaging node becomes the first node on the new route; otherwise, the first node of the current route is the source node of the data packet. In the algorithm, we do not handle the case where a data packet is salvaged, which means the nodes on the new route will not send ROUTE REPLIES if they have shorter routes to the destination. The reason is that the alternative route can be longer than the downstream nodes of the original source route. If we want to handle this case, we will use the upstream nodes in the source route header for comparison. We will use both *forward* and *backward* lists to decide whether to send a ROUTE REPLY. We consider this case as an optimization in future work.

For AODV (Ad hoc On-demand Distance Vector) [19], a source node needs to cache multiple routes to the destination to decide when to start a route discovery. Several multi-path routing protocols were proposed for AODV [36, 37]. For distance vector protocols like AODV, DRD requires a few modifications in order to prevent duplicate routes from being sent to the source.

4.5 Data flow diagram

The data flow diagram of the algorithm is shown in Fig. 4.

4.6 Discussions

We have several design decisions. First, we keep the size of the data packet with a ROUTE REQUEST small. The node sending a ROUTE REPLY adds at most three node pairs to the data packet, and an entry in a *forward* list is

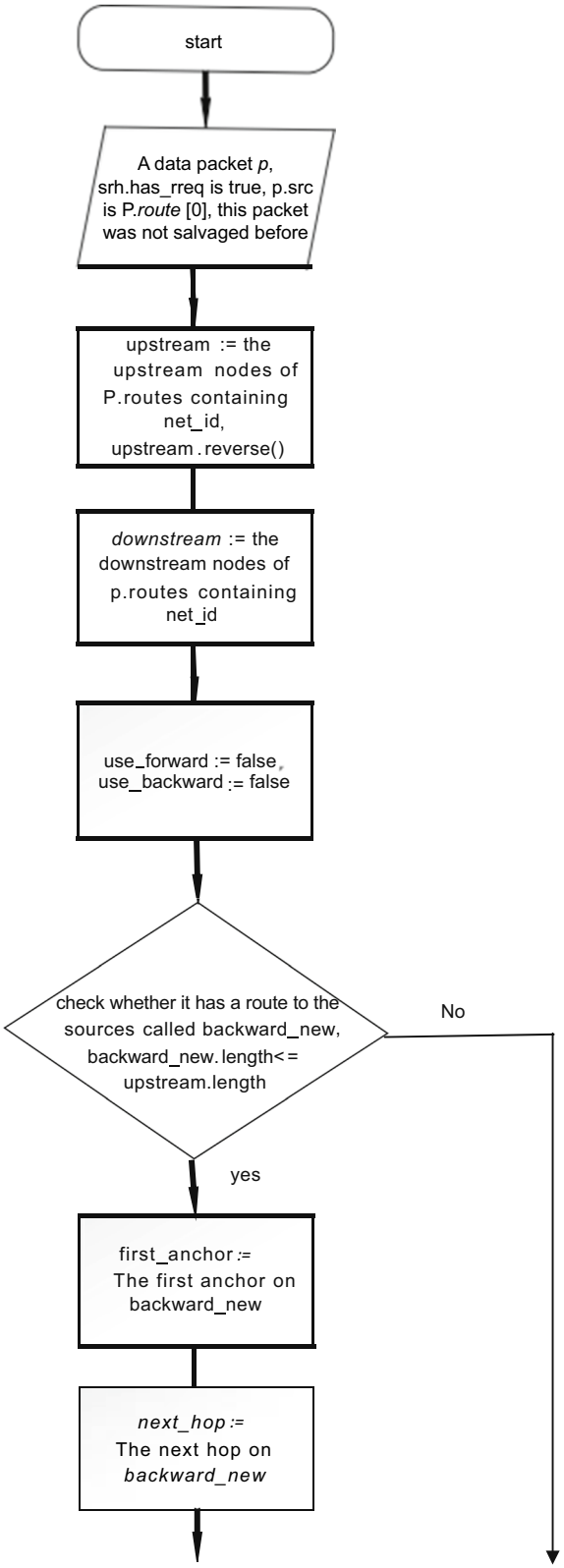


Fig. 4 The data flow diagram of the Directed Route Discovery algorithm

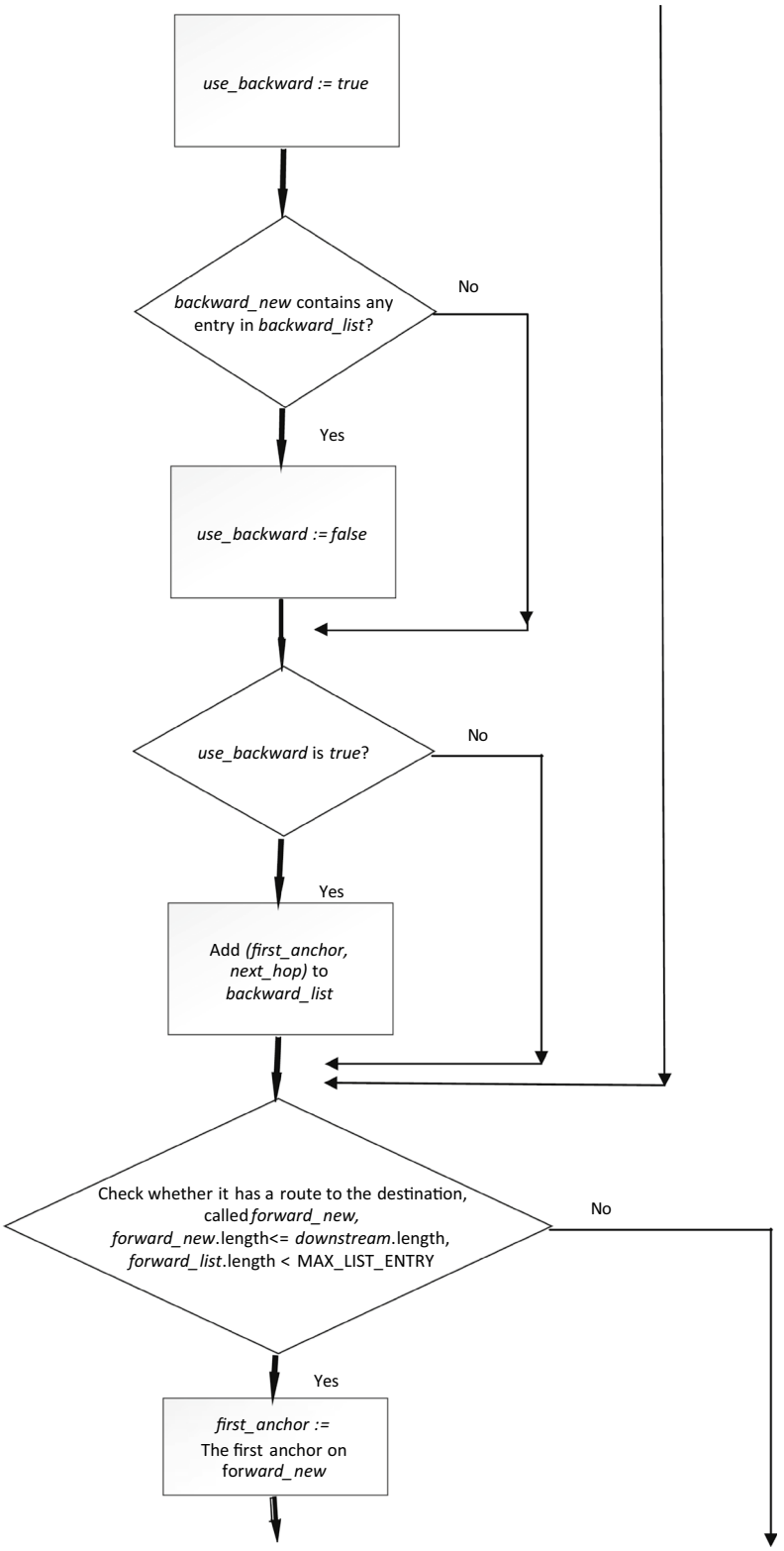


Fig. 4 (continued)

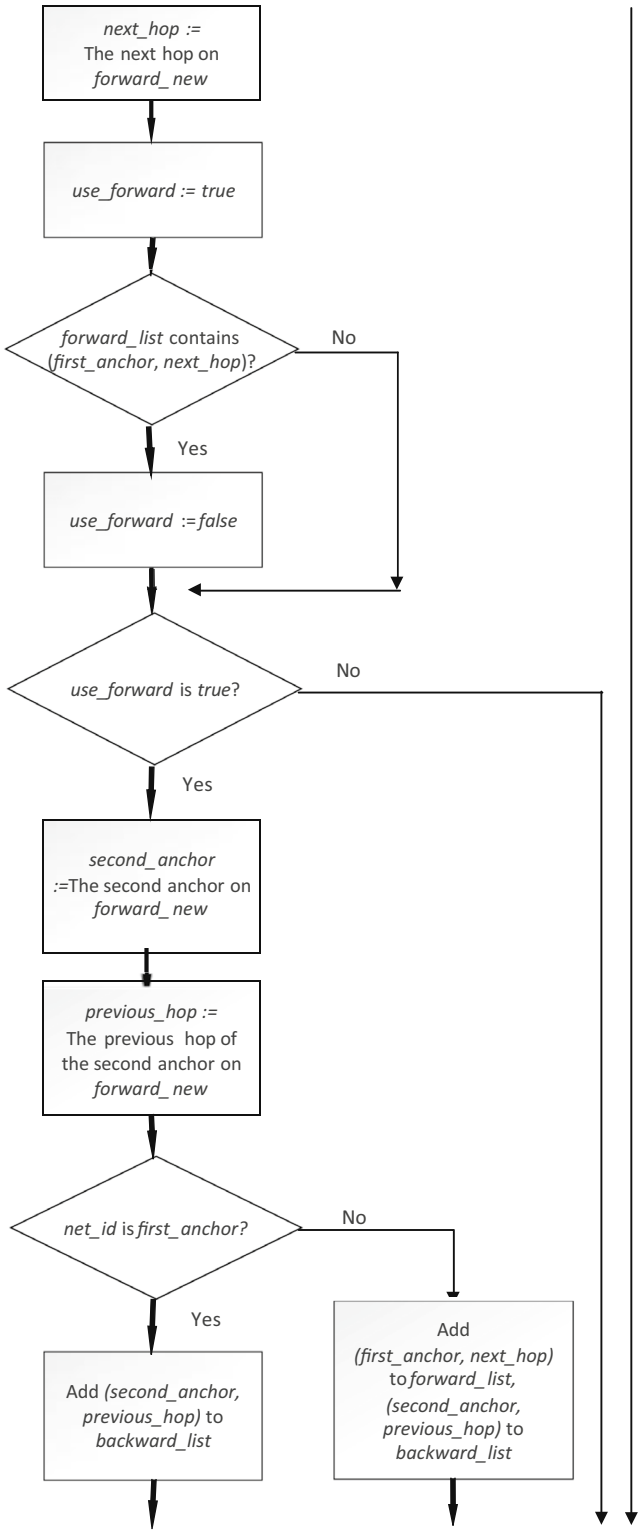


Fig. 4 (continued)

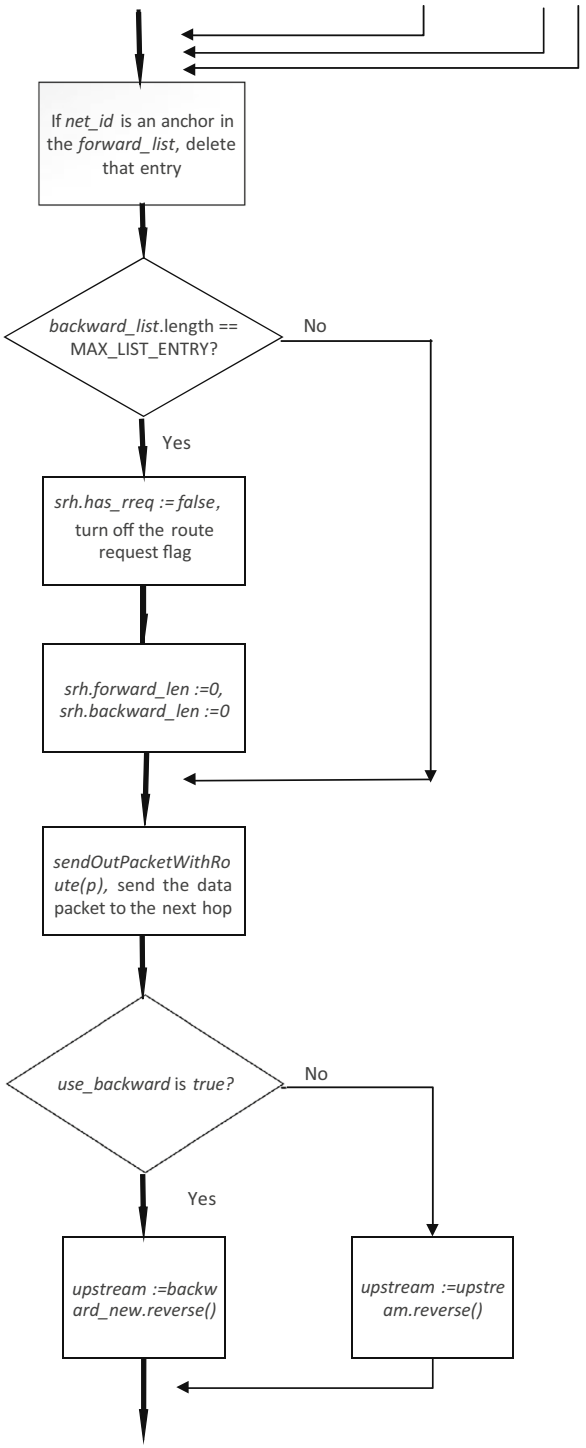


Fig. 4 (continued)

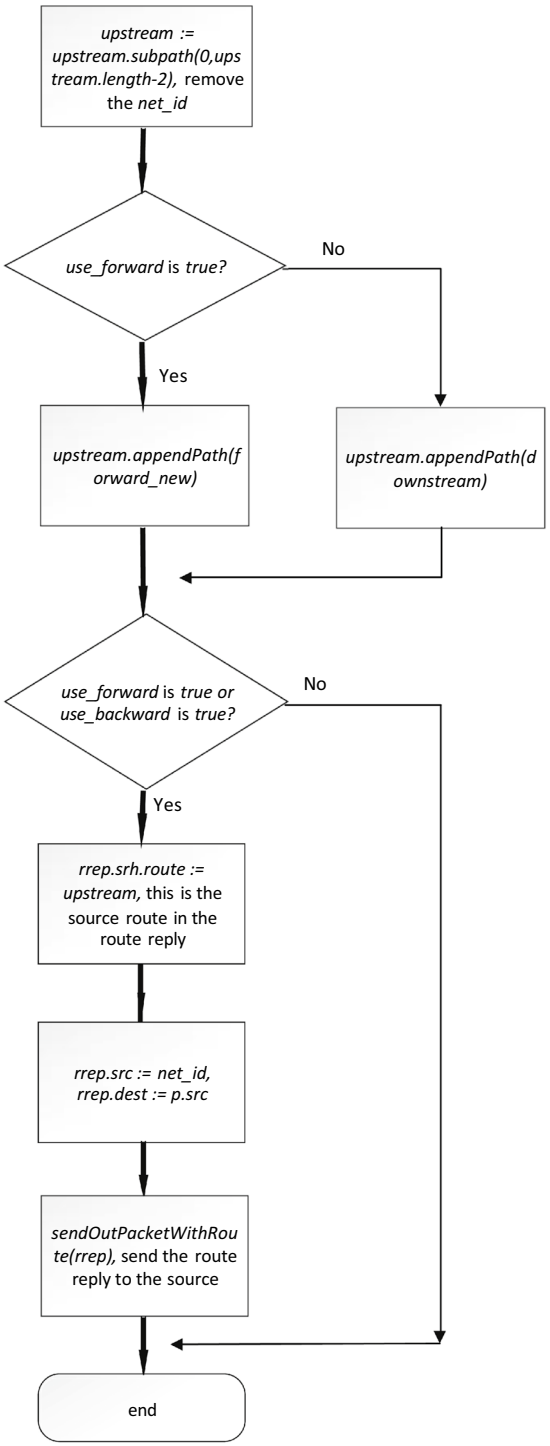


Fig. 4 (continued)

removed when an *anchor* forwards the packet. Second, a source initiates a route discovery when only one route is available in its cache. If it does so when two or more routes are available, the frequency of route discoveries will increase, and routing overhead caused by ROUTE REPLIES degrades performance. Finally, a node uses a route shorter than or having the same length as the *active* data path to send a ROUTE REPLY. Thus, DRD reduces the total size of ROUTE REPLIES. A ROUTE REPLY contains both the source route and the route to be used for sending the packet to the source.

The algorithm uses a network-wide search in several cases. First, a source did not send data packets and the last route breaks. Second, found routes and the current route have a common link, and the link breaks. Third, cached routes that satisfy the condition are not available at the intermediate nodes forwarding the data packet. Finally, no cached routes are available.

As described, a source starts the route discovery process before the last route breaks. Since ROUTE REPLIES contain the routes shorter than or having the same length as the current route, the source maintains shorter distance to the destination. Thus, both routing and data packets are localized. The two rules prevent the nodes forwarding the data packet containing the *true* flag from sending duplicate routes. ROUTE REPLIES are sent when only one route is available at the source, and the intermediate nodes send only a few ROUTE REPLIES. Thus, these control packets do not interfere data forwarding. A cached route contained in a ROUTE REPLY may be stale. Such routes are removed through on-demand Route Maintenance described in Sect. 3.

The problem of route searches is complicated, and prior work mainly focus on reducing search ranges or using probabilistic methods. There must be a completely novel approach to solve this problem. Our solution is simple, and there is no need for theoretical models. There are no ad hoc parameters in the DRD algorithm, which makes it efficient in highly mobile scenarios. Fast node movement, frequent link failures, and transmission interference, such as hidden terminals, are the major difficulties for mobile ad hoc networks; the specific tasks are route discoveries and route maintenance. Note that we explained the three reasons why the results were achieved in both Sects. 1 and 7.

5 Comparison with the SLR protocol

Sengul and Kravets [35] proposed a local recovery protocol called SLR (Source routing with Local Recovery) to reduce the frequency of ROUTE REQUEST floods. In this section, we briefly describe the protocol and our observations about SLR.

When a node detects a broken link, if it cannot salvage the packet or the packet has been salvaged, it will perform a bypass recovery. The node buffers the current packet and all packets in the network interface queue between the routing layer and the MAC layer that contain the broken link. A list of downstream nodes of these packets is

broadcast to one-hop neighbors, to see whether they are neighbors with any of those nodes. Each node maintains a MAC cache, which contains the most recent neighbor state. The node receiving the query searches its MAC cache for a neighbor listed in the query and includes all such neighbors in its reply.

When the querying node receives a reply, it repairs the packets with the broken link using the new connectivity information. One packet is marked in order to notify the destination about the broken link. The destination sends an enhanced ROUTE ERROR to the source to indicate the salvaged route as an alternative route.

We implemented this protocol and found two problems. First, SLR keeps the full path for each salvaged packet, so that the destination will send an enhanced ROUTE ERROR containing the salvaged route. A salvaged packet contains the route starting from the salvaging node. Under high mobility, a packet may be salvaged multiple times and will have a long path. We will not provide simulation results for SLR because we used high mobility rates and path length of SLR exceeds the maximum source route length in some scenarios. Second, SLR sends an enhanced ROUTE ERROR for each source and destination pair of salvaged packets, which causes higher routing overhead than DSR, such as at node mean speed of 15 m/s and 20 m/s. The authors used node speed between 0 and 20 m/s, which results in an average speed of 5 m/s [38].

The authors observed that SLR forwards more data packets over longer routes and has similar packet delivery latency as DSR, thus comparable to DSR in terms of throughput. DRD finds more shorter routes than DSR, as indicated by average data packet size, and reduces packet delivery latency by 54% at node mean speed of 20 m/s. Moreover, SLR reduces routing overhead by 20% compared with DSR, while DRD achieves between 60 and 80% reduction in routing overhead because it reduces the size of ROUTE REQUESTS and ROUTE REPLIES packets. The main difference between them is that, SLR repairs a broken link reactively while DRD proactively finds routes before the last route breaks.

6 Simulation methodology

We incorporated the algorithm into DSR with path caches. We used *ns-2.28* and *random waypoint* model [2] in which node speed was randomly chosen from $v \pm 0.1$ v m/s. Note that we did not use node speed between 0 and 20 m/s, which results in an average speed of 5 m/s [38]. We performed three sets of experiments. First, we evaluated the algorithm under various mobility rates, with node mean speed varying from 5 to 20 m/s. The highest mobility rate used in the literature is 20 m/s, which is 44.74 miles per hour and the normal driving speed in highways. Second, we evaluated how the algorithm performs as network size increases. We used node mean speed of 10 m/s and 5 m/s for the 150-node and the 200-node scenarios, respectively. We used 5 m/s for the latter because small cache size in DSR causes a large amount of route discoveries under high mobility and large networks. Using lower speed reduces the effect of cache size on the results. Finally, we evaluated it using different loads, such as 10 flows, 20 flows, 30 flows and 40 flows, in order to see how network load affects it.

We used CBR (Constant Bit Rate) traffic with 4 packets per second and packet size of 64 bytes as in [2]. The sources and destinations were chosen randomly. We used the same evaluation method as in [2] and [32]. The way the source node and the destination node are chosen is irrelevant to the performance of DRD. The route discovery process starts from broadcastings or network-wide flooding. The source node and the destination node move randomly and can be in any locations during the simulations. We used four field configurations: a 1500 m × 500 m field with 50 nodes, a 2200 m × 600 m field with 100 nodes [2], a 2200 m × 1000 m field with 150 nodes, and a 2200 m × 1200 m field with 200 nodes. The field sizes were chosen to keep similar node density for different network sizes. Simulations ran for 900 s. Each data point represents an average of 10 runs of randomly generated scenarios. The results will be shown with the error bars in the graphs representing the 95% confidence interval of the average. We used the following metrics in the evaluations:

- Packet delivery ratio: the ratio between the number of packets received by the CBR sink at the destination and the number of packets originated by the CBR source.
- Packet delivery latency: the delay from when a data packet is sent until it is received by the destination.
- Route discovery overhead: the total size of ROUTE REQUESTS and ROUTE REPLIES sent and forwarded.
- Routing overhead: the total size of ROUTE REQUESTS, ROUTE REPLIES and ROUTE ERRORS sent and forwarded.
- Route requests sent: the total number of ROUTE REQUESTS sent by the sources, which is the total number of network-wide route searches. Our main goal is to reduce this metric, and the second goal is to reduce route discovery overhead.
- Average data packet size: the total size of data packets received by the destination divided by the total number of data packets received by the destination.
- Packet overhead: the total number of ROUTE REQUESTS, ROUTE REPLIES and ROUTE ERRORS sent and forwarded.

- Route errors sent and forwarded: the total number of ROUTE ERRORS sent and forwarded. DSR piggybacks the last broken link information to the next ROUTE REQUEST, and thus this metric also reflects the amount of ROUTE REQUESTS broadcast in the network.

7 Simulation results

7.1 Packet delivery ratio

Figure 5(a), (c), (e), and (g) show packet delivery ratio. For the 100-node scenarios, DRD outperforms DSR by 15% at node mean speed of 20 m/s. The improvement increases as mobility increases because of the reduction in routing overhead, which will be described shortly. As mobility increases, routes break more frequently, and network-wide route searches cause more MAC collisions and packet losses. For the 50-node scenarios, DRD has similar performance as DSR because for small networks, route searches do not incur much routing overhead. DRD outperforms DSR by 7% for the 150-node scenarios at node pause time of 0 s.

7.2 Packet delivery latency

Figure 5(b), (d), (f), and (h) show packet delivery latency. DRD has higher latency than DSR by 0.008 s for 50-node scenarios at node mean speed of 10 m/s. This phenomenon may be caused by the higher number of ROUTE REPLIES. For the 100-node scenarios, DRD reduces packet delivery latency by 54% at node mean speed of 20 m/s. DRD reduces route acquisition latency because it discovers routes before the last route breaks. Moreover, it discovers the routes shorter than the current data path. As nodes move and actively cache overheard routes, many short routes around the data path become available. DRD discovers these routes in a timely manner, thus significantly reducing packet delivery latency. For example, it reduces latency by 48% and 45% for the 150-node and the 200-node scenarios, respectively. The reduction in packet delivery latency increases as mobility increases.

7.3 Route discovery overhead and routing overhead

Figure 6(a), (c), (e), and (g) show route discovery overhead. DRD reduces route discovery overhead by 57% and 71% for the 50-node and the 100-node scenarios, respectively. For the 100- node scenarios, DRD reduces the size of ROUTE REQUESTS by 90% and the size of ROUTE REPLIES by 74%, as shown in Fig. 7. Intermediate nodes send ROUTE REPLIES containing routes shorter or with the same length as the data path; therefore, the size of ROUTE REPLIES is reduced. The smaller size of ROUTE REQUESTS indicates that most ROUTE REQUESTS are forwarded by nodes not far away from the source. Since a source discovers shorter routes, the nodes close to the source are in the vicinity of the destination and have cached routes to the destination. Due to the locality of routing packets, route discovery overhead increases slowly as mobility increases. For the 200-node scenarios, DRD reduces route discovery overhead by 81%.

Figure 6(b), (d), (f), and (h) show the total size of routing packets, namely routing overhead. Routing overhead is the total size of ROUTE REQUESTS, ROUTE REPLIES, and ROUTE ERRORS. DRD reduces routing overhead by 52% for the 50- node scenarios and 77% for the 100-node scenarios. DRD reduces routing overhead by 78% for both the 150-node and 200-node scenarios. Since it discovers shorter routes as nodes move, the paths traversed by ROUTE ERRORS are shorter too, compared with ROUTE ERRORS sent when network-wide searches are used. The size of ROUTE ERRORS depends on the paths used to send ROUTE ERRORS. Routing overhead remains low as mobility increases. Thus, DRD makes DSR scalable with respect to mobility, since routing overhead is the measure of scalability used in the literature.

7.4 Route requests sent and average data packet size

Figure 8(a), (c), (e), and (g) show the number of ROUTE RE QUESTS sent. As long as an intermediate node has a cached route to the destination, DRD reduces a network-wide search. Normally several distinct routes will be sent to the source. The source switches to the shortest one and starts the next route discovery when one route is left in its cache. If no ROUTE REPLIES are sent to the source, it uses flooding to find routes. For the 100-node scenarios, DRD reduces route searches by 17%. For the 150-node and 200-node scenarios, DRD reduces route searches by 12% and 17%, respectively. The algorithm reduces the total size of ROUTE REQUESTS by 75% for the 150-node scenarios and 79% for the 200-node scenarios, as shown in Fig. 7. Recall that a ROUTE REQUEST contains a sequence of nodes traversed by the packet. Thus, ROUTE REQUESTS sent through broadcast are restricted to the nodes close to a source.

We measured average data packet size and show the results in Fig. 8(b), (d), (f), and (h). The average data packet size of DRD is smaller than that of DSR. Although node pairs in the source route header of a data packet increase packet size, only the data packet containing *true* for *has_rreq* has this overhead. The smaller data packet size demonstrates that DRD finds more shorter routes.

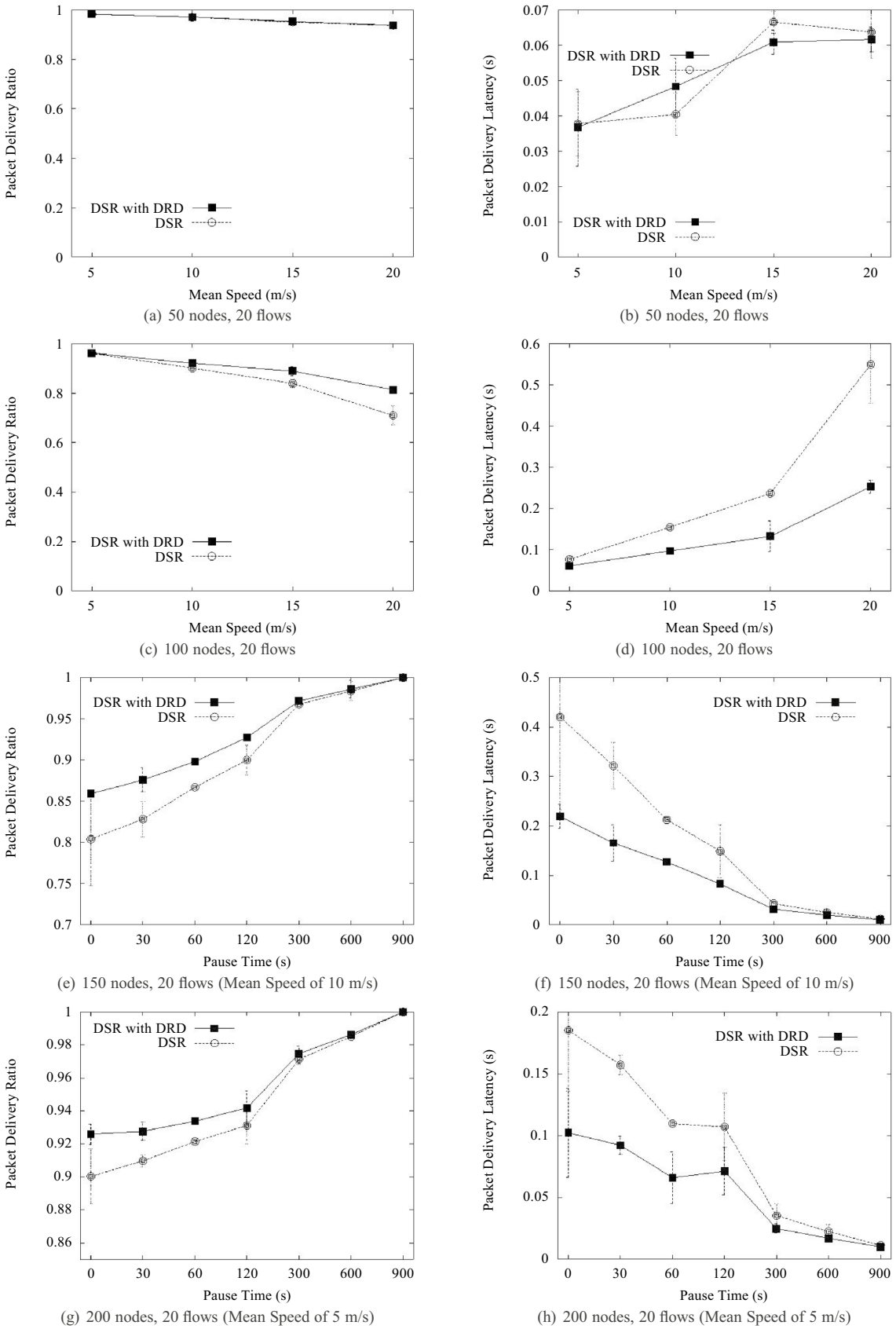


Fig. 5 Packet Delivery Ratio and Packet Delivery Latency

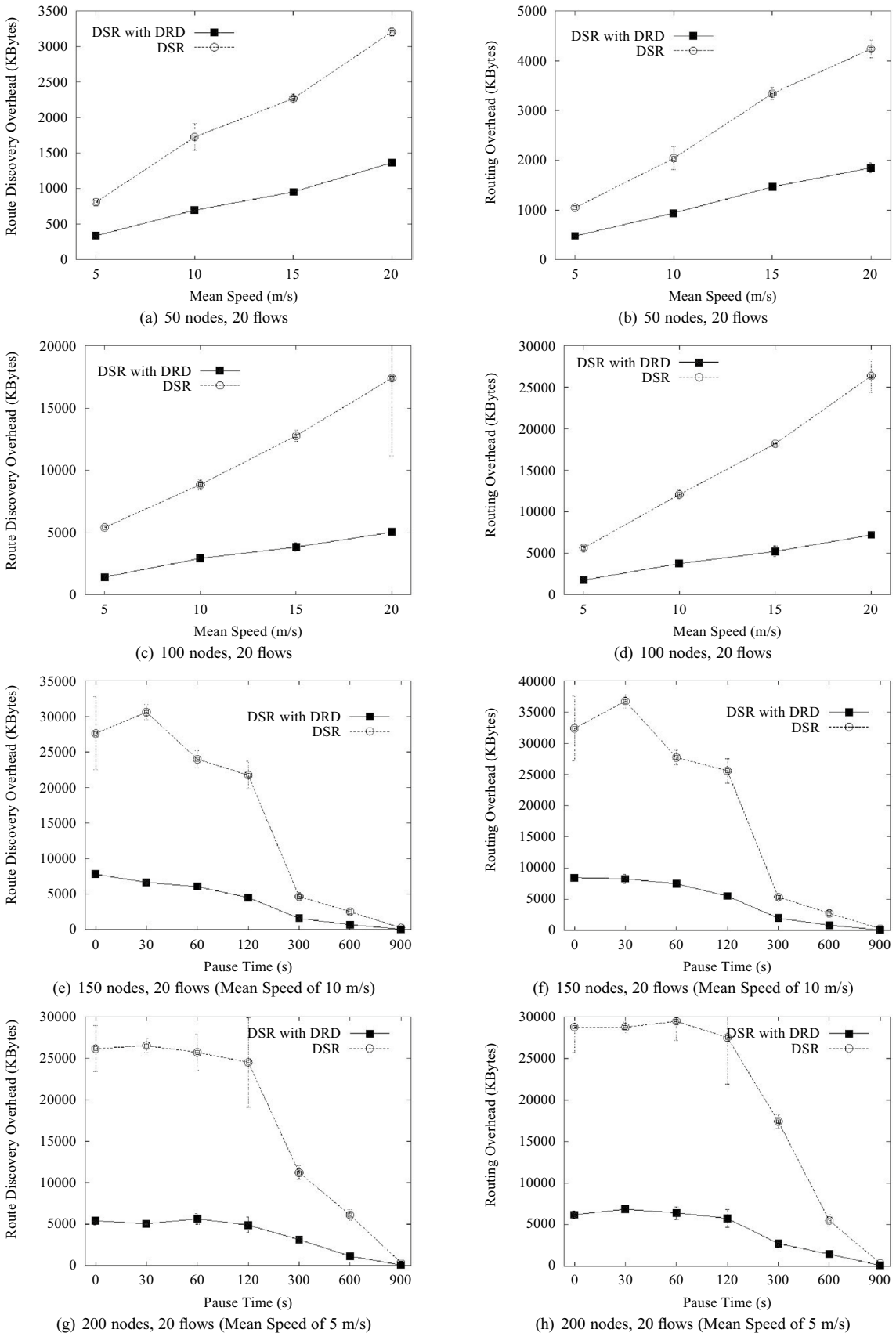


Fig. 6 Route Discovery Overhead and Routing Overhead

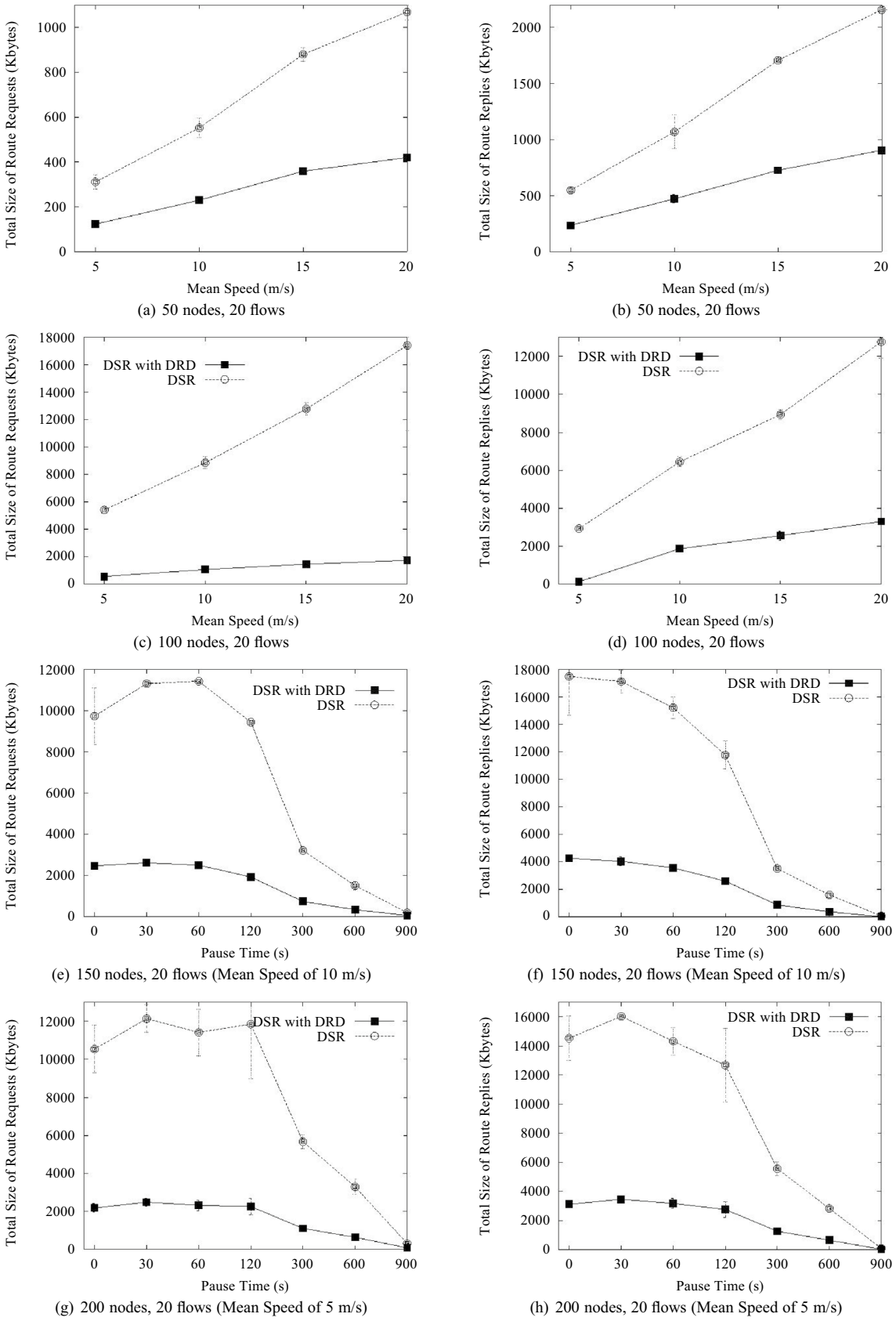


Fig. 7 Total Size of Route Requests and Route Replies

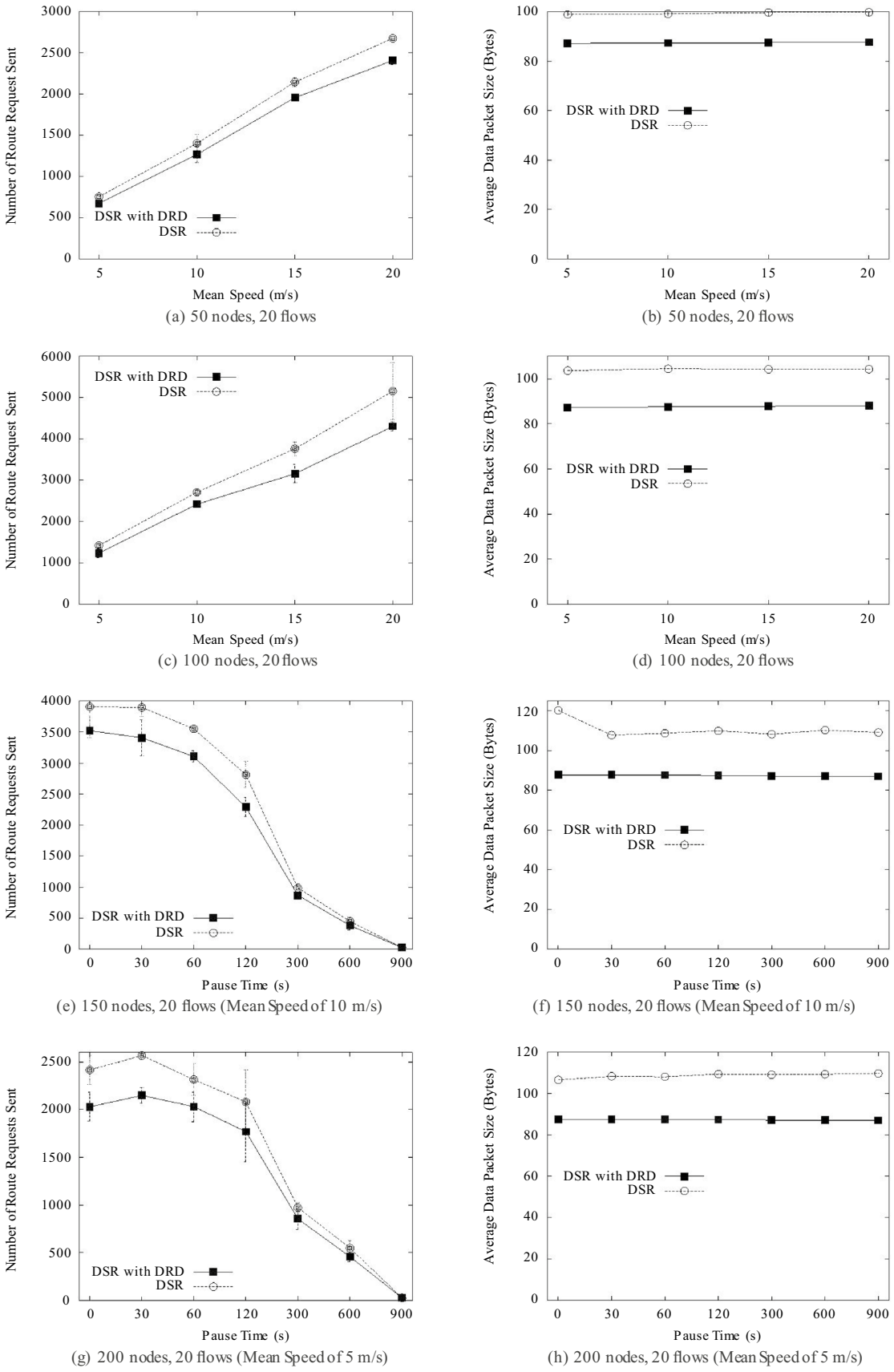
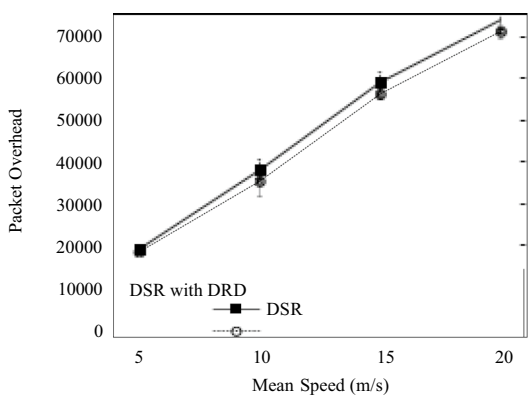
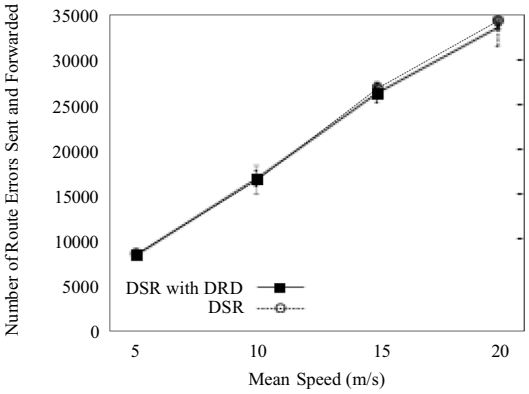


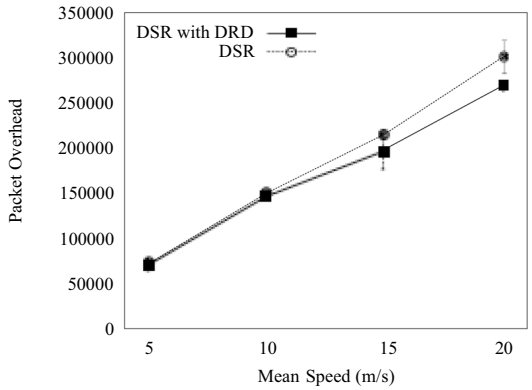
Fig. 8 The Number of Route Requests Sent and Average Data Packet Size



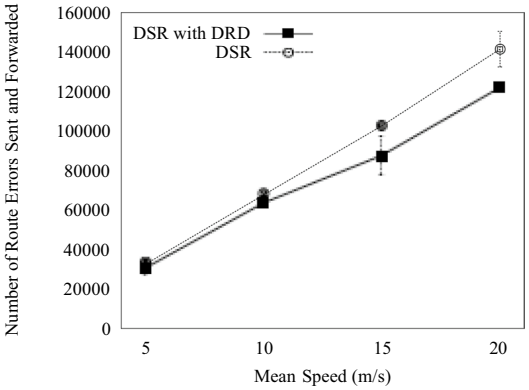
(a) 50 nodes, 20 flows



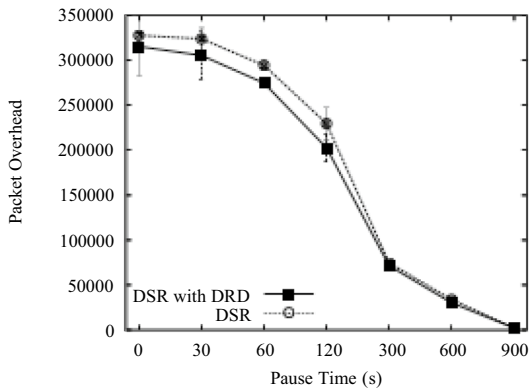
(b) 50 nodes, 20 flows



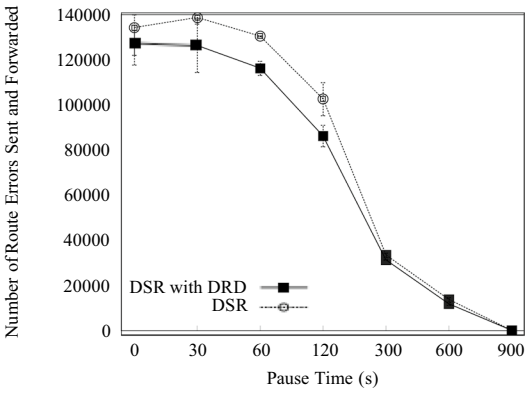
(c) 100 nodes, 20 flows



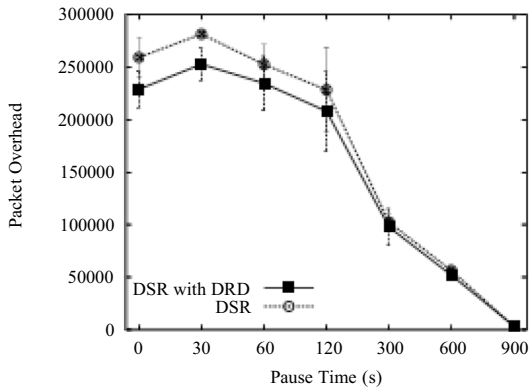
(d) 100 nodes, 20 flows



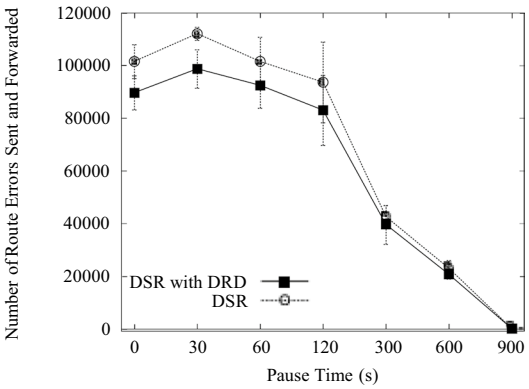
(e) 150 nodes, 20 flows (Mean Speed of 10 m/s)



(f) 150 nodes, 20 flows (Mean Speed of 10 m/s)



(g) 200 nodes, 20 flows (Mean Speed of 5 m/s)



(h) 200 nodes, 20 flows (Mean Speed of 5 m/s)

Fig. 9 Packet Overhead and the Number of Route Errors Sent and Forwarded

7.5 Packet overhead and route errors generated

Figure 9 shows packet overhead and the number of ROUTE ERRORS sent and forwarded. As shown in Fig. 9(c), DRD reduces packet overhead by 17% at node mean speed of 20 m/s. The reduction increases as mobility increases because more link failures occur and source nodes need to find routes more often. As shown in Fig. 9(e) and (g), DRD reduces packet overhead by up to 8% and 15% for the 150-node and 200-node networks at pause time of 30 s, respectively. As shown in Fig. 9(d) and (h), DRD reduces the number of ROUTE ERRORS by up to 14% and 15% for the 100-node and 200-node networks at node mean speed of 20 m/s and pause time of 30 s, respectively. When a source starts a route discovery, it will piggyback the last broken link information to the ROUTE REQUEST, and thus such ROUTE REQUESTS are counted as ROUTE ERRORS.

7.6 The scalability of DRD as network load increases

Figure 10 shows the performance of the algorithm as network load increases. The maximum load is 40 CBR flows. DRD improves packet delivery ratio by 22% and reduces packet delivery latency by 46% compared with DSR. It reduces both route discovery overhead and routing overhead by 72%. As network load increases, more routes break because of serious wireless interference and packet collisions. A link breaks if the sending node does not receive an acknowledgement from the next hop after the maximum number of retransmissions, which is seven for IEEE 802.11. Thus, DSR initiates more network-wide route searches for high load scenarios. For DRD, routing overhead increases slowly as network load increases, which indicates that it is robust to network load.

We believe this result is significant and helps us understand the operation of the algorithm better. The average data packet size is 104.5, 104.4, 103.4, and 101 bytes for DSR, and is 87.5 bytes for DRD as network load increases. The smaller packet size of DRD shows that DRD finds shorter routes even under high load. However, for both DSR and DRD, packet delivery ratio and packet delivery latency are unfavorable for the 40-flow scenarios because of a large amount of wireless interference and packet losses.

8 Conclusions

In this paper, we presented an algorithm called DRD (Directed Route Discovery) that exploits data transmission for route discoveries. When a source has only one route to a destination, it sets a boolean variable in a data packet to be *true*, which indicates that it needs routes to the destination. This variable is a new form of ROUTE REQUEST. The nodes forwarding the data packet send ROUTE REPLIES to the source using cached routes. To prevent nodes from sending duplicate routes, we define a *forward* list and a *backward* list to record route diverging and converging information about the cached route in a ROUTE REPLY. Subsequent nodes use this information to decide whether to send a ROUTE REPLY to the source node. The route contained in a ROUTE REPLY is shorter than or has the same length as the *active* data path from the source to the destination. Thus, our algorithm reduces packet delivery latency and the total size of ROUTE REPLIES.

We show that the algorithm significantly improves packet delivery ratio and reduces packet delivery latency. For example, it improves packet delivery ratio by 15% and reduces latency by 54% for the 100-node networks at node mean speed of 20 m/s. Packet delivery latency consists of route discovery latency and end-to-end delivery latency, which is determined by hop count or route length. The algorithm reduces route discovery latency because it finds routes before the last route breaks, and discovers routes shorter than the *active* data path. It also significantly reduces network-wide route searches and the total size of ROUTE REQUESTS and ROUTE REPLIES. For the 100-node networks, DRD reduces route searches by 17%. For the 150-node and 200-node networks, DRD reduces route searches by 12% and 17%, respectively. For the 100-node networks, DRD reduces the size of ROUTE REQUESTS by 90% and the size of ROUTE REPLIES by 74%. Note that route discovery overhead is the total size of ROUTE REQUESTS and ROUTE REPLIES. Finally, routing overhead increases slowly as mobility or network load increases. DRD achieves between 52 and 78% reduction in routing overhead for the 50-node, 100-node, 150-node and 200-node networks. This particular result demonstrates that it is independent of mobility.

The type of networks we study is mobile ad hoc networks where nodes move randomly and topology changes are unpredictable. Most previous studies used a maximum node speed of 20 m/s (average speed of 10 m/s) in simulations. In contrast, we used mean node speed ranging between 5 and 20 m/s, which is 44.74 miles per hour and the normal driving speed in highways. Therefore, our algorithm works in highly mobile scenarios. Examples of such networks include battlefields and networks containing vehicles or spacecrafts, such as vehicular ad hoc networks and space networks. DRD reduces the average distance traversed by ROUTE REQUESTS and ROUTE REPLIES, and routing packets are localized between the source and the destination. Thus, it significantly reduces transmissions and the interference to nearby nodes; improving network throughput under high mobility is hard because mobility causes frequent route failures. The key feature of the algorithm is that it tracks the latest location of the destination when it moves through an *active* data connection. This work applies to larger mobile ad hoc networks with more than 200 nodes and sensor networks where flooding is commonly used for *downward* traffic to

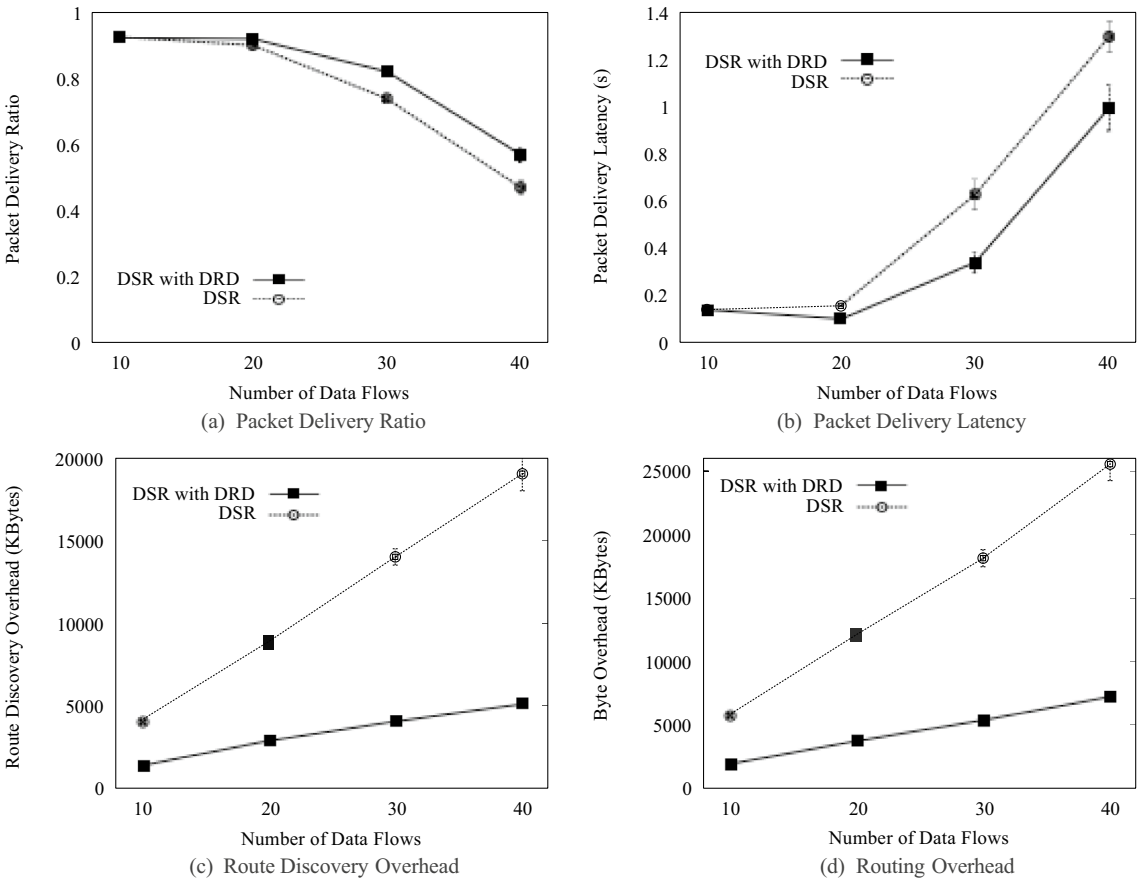


Fig. 10 Under Dynamic Network Load (100 nodes, Mean Speed of 10 m/s, Pause Time 0 s)

monitored areas with RPL for LLNs (Low-power and Lossy Networks).

This work also contributes to the understanding of the scalability of DSR, or on-demand routing protocols. We solve the route discovery problem of DSR by reducing the number of network-wide searches and the total size of route discovery packets. Our algorithm reduces route discovery overhead significantly under high mobility and network load. The results for high load are shown in Fig. 10. Thus, we solve the traditional broadcast storm problem, because the algorithm reduces broadcastings by finding routes nearby the *active* data path. We believe that broadcasting is still required and cannot be avoided completely. One future direction is to study how DRD performs in sensor networks and how much energy it can save for battery-powered small devices. It was well-known that transmissions consume more energy than computations in the early 90 s. We conclude that it is important to reduce the total number of route searches in order to reduce transmissions and wireless interference.

Acknowledgements The author would like to thank the anonymous reviewers for their careful and detailed comments, which help improve the presentation of this paper.

References

1. Ni, S., Tseng, Y., Chen, Y., & Sheu, J. (1999). The broadcast storm problem in a mobile ad hoc network. In *Proceedings of ACM MobiCom*.
2. Broch, J., Maltz, D., Johnson, D., Hu, Y.-C., & Jetcheva, J. (1998). A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of ACM MobiCom* (pp. 85–97).
3. Ko, Y.-B., & Vaidya, N. (2000). Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, *6*
4. Castaneda, R., & Das, S. (1999). Query localization techniques for on-demand routing protocols in ad hoc networks. In *Proceedings of ACM MobiCom* (pp. 186–194).
5. Haas, Z., Halpern, J., & Li, L. (2002). Gossip-based ad hoc routing. In *Proceedings of IEEE INFOCOM*.
6. Ferriere, H., Grossglauser, M., & Vetterli, M. (2003). Age matters: Efficient route discovery in mobile ad hoc networks using encounter ages. In *Proceedings of ACMMobiHoc* (pp. 257–266).
7. Beraldi, R. (2008). The polarized gossip protocol for path discovery in MANETs. *Ad Hoc Networks, 6*(1), 79–91.
8. Johnson, D., Maltz, D., & Hu, Y.-C. (2004). The dynamic source routing for mobile ad hoc networks, IETF Internet Draft.
9. Li, J.-Y., Blake, C., Couto, D., Lee, H., & Morris, R. (2001). Capacity of ad hoc wireless networks. In *Proceedings of ACM MobiCom*.
10. Grimmett, G. (1989). *Percolation*. Springer-Verlag.
11. Hui, P., Crowcroft, J., & Yoneki, E. (2011). Bubble rap: social based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing, 10*, 1576–1589.

12. Fall, K. (2003). A delay-tolerant network architecture for challenged Internets. In *Proceedings of ACM SIGCOMM*.
13. Lee, J. W., Kusy, B., Shihada, B., Azim, T., & Levis, P. (2010). Whirlpool routing for mobility. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing(MobiHoc)*.
14. Ladas, A., Pavlatos, N., Weerasinghe, N., & Politis, C. (2016). Multipath routing approach to enhance resiliency and scalability in ad-hoc networks. In *Proceedings of IEEE International Conference on Communications (ICC)*.
15. Ramrekha, A., & Politis, C. (2010). A hybrid adaptive routing protocol for extreme emergency ad hoc communication. In *Proceedings of 19th International Conference on Computer Communications and Networks(ICCCN)*.
16. Tsirigos, A., & Haas, Z. (2001). Multipath routing in the presence of frequent topological changes. In *IEEE Communication Magazine* (pp. 132–138).
17. Nasipuri, A., Castaneda, R., & Das, S. (2001). Performance of multipath routing for on-demand protocols in mobile ad hoc networks. In *Mobile Networks and Applications* (pp. 339–349).
18. Jacquet, P., Muhlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., & Viennot, L. (2001). Optimized link state routing protocol for ad hoc networks. In: *Proceedings of IEEE International Multi Topic Conference (INMIC), Technology for the 21st Century* (pp. 62–68).
19. Perkins, C. E., & Royer, E. M. (1999). Ad hoc on-demand distance vector routing. In *Proceedings of IEEE Workshop on Mobile Computing System and Applications (WMCSA)* (pp. 90–100).
20. De Couto, D., Aguayo, D., Bicket, J., & Morris, R. (2003). A high-throughput path metric for multi-hop wireless routing. In *Proceedings of ACM MobiCom* (pp. 134–146).
21. Tang, L., Sun, Y. J., Gurewitz, O., & Johnson, D. B. (2012). Optimizations for route discovery in asynchronous duty-cycling wireless networks. In *Proceedings of IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*.
22. Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J. P., & Alexander, R. (2012). RPL: IPv6 routing protocol for low-power and lossy networks. In *RFC6550, IETF*.
23. Istomin, T., Iova, O., Picco, G., & Kiraly, C. (2020). Route or flood? reliable and efficient support for downward traffic in RPL. *IEEE Transactions on Sensor Networks, 16*(1), 1–41.
24. Zakarya, M., Khan, A. A., Qazani, M., Ali, H., A.-Bahri, M., Khan, A., Ali, A., & Khan, R. (2024). Sustainable computing across datacenters: A review of enabling models and techniques. *Elsevier Computer Science Review*.
25. Araniti, G., Bezirgiannidis, N., Birrane, E., Bisio, I., Burleigh, S., Caini, C., Feldmann, M., Marchese, M., Segui, J., & Suzuki, K. (2015). Contact graph routing in DTN space networks: overview, enhancements and performance. *IEEE Communications Magazine, 53*(3), 38–46.
26. Madoery, P., Raverta, F., Fraire, J., & Finochietto, J. (2018). Routing in space delay tolerant networks under uncertain contact plans. In *Proceedings of IEEE International Conference on Communications(ICC)*.
27. Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., & Weiss, H. (2007). Delay-tolerant networking architecture. In *IETF RFC 4838, Informational*.
28. Wyatt, J., Burleigh, S., Jones, R., Torgerson, L., & Wissler, S. (2009). Disruption tolerant networking flight validation experiment on NASA's EPOXI mission. In *Proceedings of IEEE First International Conference on Advances in Satellite and Space Communications*.
29. Madoery, P., Fraire, J., Raverta, F., Finochietto, J., & Burleigh, S. (2018). Managing routing scalability in space DTNs. In *Proceedings of IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*.
30. Ko, Y.-B., & Vaidya, N. (1998). Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of ACM MobiCom* (pp. 66–75).
31. Hui, P., Crowcroft, J., & Yoneki, E. (2008). Bubble rap: social based forwarding in delay tolerant networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing(MobiHoc)*.
32. Hu, Y.-C., & Johnson, D. (2000). Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *Proceedings of ACM MobiCom* (pp. 231–242).
33. Yu, X., & Kedem, Z. (2005). A distributed adaptive cache update algorithm for the dynamic source routing protocol. In *Proceedings of IEEE INFOCOM*.
34. Yu, X. (2005). Mobility, route caching and TCP performance in mobile ad hoc networks. *Ph.D. thesis, New York University*.
35. Sengul, C., & Kravets, R. (2006). Bypass routing: an on-demand local recovery protocol for ad hoc networks. *Ad Hoc Networks, 4*(3), 380–397.
36. Lee, S.-J., & Gerla, M. (2000). AODV-BR: backup routing in ad hoc networks. In *Proceedings of IEEE WCNC*.
37. Marina, M., & Das, S. (2001). On-demand multipath distance vector routing in ad hoc networks. In *Proceedings of IEEE ICNP*.
38. Yoon, J., Liu, M., & Noble, B. (2006). A general framework to construct stationary mobility models for the simulation of mobile networks. *IEEE Transactions on Mobile Computing*, *5*(7).

Xin Yu received her B.E. degree in electrical engineering from the Shandong Polytechnic University (It became a part of Shandong University in July 2000), China, in July 1995, and her M.E. degree in computer engineering from Institute of Automation, Chinese Academy of Sciences, Beijing, in July 1998. She joined New York University in August 2000 and received her M.S. and Ph.D. degrees in computer science in September 2002 and May 2005, respectively. She was a postdoctoral researcher at Thomson Corporate Research (Technicolor), Princeton, New Jersey from 2005 to 2006 working on wireless mesh networks and IEEE 802.11s related routing proposals. She was an associate at Blackrock, New York city from 2007 to 2009 working on asset management. She worked at Cisco Systems, Milpitas, California from 2011 to 2014 on Video Surveillance Systems in Physical Security Business Unit (PSBU), WAN optimizations (Encrypted-MAPI protocols), and Cloud Service Router in the Service Routing Task Group (SRTG), collaborating with colleagues from Networks and Operating Systems Task Group (NOSTG) on onePK. She is currently working at Sen Pu Software Corporation, Jinan, China. Her research interests include mobile ad hoc networks, sensor networks, and wireless mesh networks, mainly focusing on routing issues.